JOURNAL OF HIGH-FREQUENCY COMMUNICATION TECHNOLOGIES
Vol. 01 (04) (2023), pp.111-119. ***e-ISSN:2836-9009***
https://doi.org/10.58399/PSPD6084

Hand Cricket Game Application Using Computer Vision

Kaythry P[1*], Jegadish Kumar K J[2], Vinu R[3], Balaji M[4], M Errshaad Ahamed[5]

[1,2,4,5] Department of Electronics and Communication Engineering, Sri Sivasubramaniya Nadar College of Engineering, Chennai, Tamil Nadu, India *E-mail address: kaythryp@ssn.edu.in, jegadishkj@ssn.edu.in, balaji.murugaiya@gmail.com, errshyssn@gmail.com*
[3]Department of Electronics and Communication Engineering, Dayananda Sagar University, Bengaluru, India *E-mail address: vinur-ece@dsu.edu.in*

Abstract

Hand cricket is an attempt to experience the childhood entrainment of playing cricket using our hands. This paper presents an application built to play Hand cricket using a computer vision-based real-time 3D hand gesture recognition system. To play the game the user will give inputs using hand gestures. These hand gestures will be identified using a real-time computer vision system that runs based on a CNN model. Since in this game both players must play simultaneously the computer also comes up with a number from 1 to 6 when a hand gesture is being read and recognized. The core engine of building this app is an image classifier that classifies a picture of a hand into 1,2,3,4,5,6 and none. This paper shows an interactive simple game application developed, that tries to make the game feel more natural and as if playing with a real person.

Keywords: *Computer Vision, Hand Cricket, Web Application, Machine Learning Model, Game*

1. Introduction

Hand cricket is a simple yet amazing game to play with your friends and family. Although the game is quite simple, it can extend for a very long duration too. Hand cricket is indeed the most sorted playful activity by school-going kids. It rejuvenates the players to the core and helps them in killing the pangs of boredom. It is the best to play to get rejuvenation and bring in excitement. Two people can enjoy it to the fullest without the need for additional equipment and arrangements. Cricket is indisputably a king of sports, but we can regard hand cricket as the king of leisure games. Hand cricket like that of Rock-Paper-Scissors. In the game, there are 2 players, and they get to bat or bowl according to the chance. Both players get to make hand signs from 1 to 6. The player who bats gets to score runs by showing hand signs and according to the signs their score adds up and the player is out. The other player gets to bat and whether he beats the previous player's score or not determines the winner.

The already existing web applications hand cricket, require the user to enter the score manually. To avoid this, we developed a web application that calculates the score of the player using hand gestures made by the user. This type of live interaction between the user and the computer will attract more users. In the game, an application developed the user plays with the computer and it

[*]**Corresponding author**
Received: 11^{th}, July 2023, Revised: 14^{th} September, 2023, Accepted: 25^{th} October 2023 and available online 19^{th} November 2023

This work is licensed under a Creative Commons Attribution 4.0 License(CC BY 4.0).
For more information, see https://creativecommons.org/licenses/by/4.0/

comes up with random hand signs. The key challenge here would be for a computer to recognize the hand gesture from real-time images (video/webcam feed) and calculate the score using the user's move and the computer's move. The image of different hand gestures is shown in Fig. 1.

The initial step taken is to come up with a dataset to train the model. To come up with the dataset for the hand signs required for the game, manually 1400 images were taken over 2 weeks in different surroundings. The dataset contains images belonging to 7 classes, depicting 1, 2, 3,4,5,6, and none. Once the data set is created, a model is made to train our dataset and make the classification. In this model project, SqueezeNet architecture was used to create the model. It was used to get an accurate model that did not take much time to make the classification.

The trained model was then deployed into the open Computer Vision (CV) application. The application is designed in such a way that the camera feed is on throughout the run time and when the hand gesture changes, the deployed model makes a prediction. By comparing the prediction and the random computer move generated the application updates the score and determines a winner. The rest of the paper is organized as follows: Section 2 discusses hand gesture recognition-related works in the literature. The proposed work using computer vision is given in detail in Section 3. In Section 4, the procedure for building the game is presented. The results and discussions of the work are given in Section 5. Section 6 of the paper concludes the work.

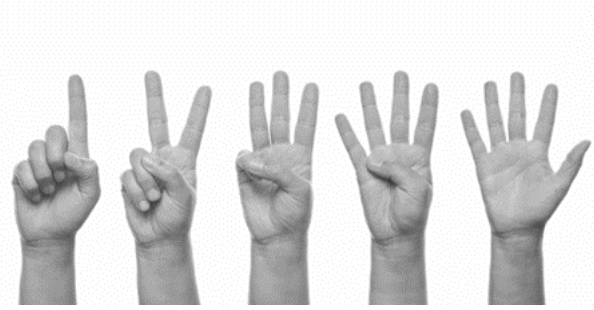

Figure 1. Image of different hand gestures

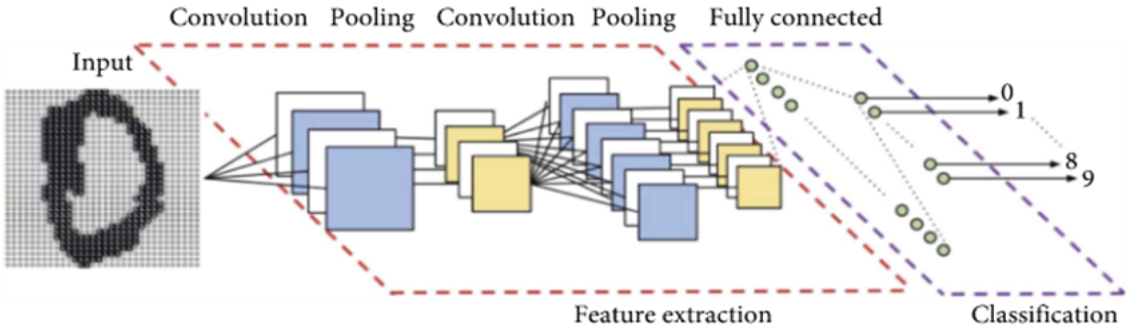

Figure 2. Structure of CNN model

2. Related Work

Recently Convolutional Neural Network (CNN) has become the most evident method for image classification. Also, different research in the literature has exhibited the need of network architecture to achieve high performance by making changes in the different layers. Hand movement and gestures are recognized using mathematical algorithms for transfer learning for Human-Computer Interaction (HCI). The gestures are trained to perform actions such as scrolling and switching pages B. et al. (2020) & Triviño-López et al. (2020). According to Prakash and Gautam (2019) the hand gesture recognition enables the machine to follow commands by identifying the different angle movements of the hand. A simple and computationally efficient, gradient-based optimization algorithm is proposed Kingma and Ba (2015). The authors targeted machine learning problems

Figure 3. A sample dataset used

with large datasets and/or high-dimensional parameter spaces. Their proposed method combines the advantages of Adagard to deal with sparse gradients and Root Mean Square Propagation (RMSProp) to deal with sparse gradients. The experiments confirm that the combination of optimization methods was robust and well-suited to a wide range of nonconvex optimization problems in the field of machine learning. The importance of preprocessing techniques for image classification with the CIFAR10 dataset, for three different variations of convolutional neural networks are carried out Sudeep and Pal (2017). The results, zero component analysis (ZCA) outperformed both the conventional mean normalization and standardization techniques for all the three variations of CNNs. Munasinghe developed a computer vision and neural network-based system to recognize real-time gestures using motion history images (MIH) Munasinghe (2018). The experiment results prove that the motion history images with the application of neural network provide a satisfactory result.

By referring to the unique approach for scaling hand gestures such that CNN can identify them without requiring an enormous quantity of training data or extratraining effort Chen et al. (2014). The network architecture proposed uses the gesture tuples with a novel Viterbi-like decoder. This work experiments on 2D and 3D versions of the Squeeze Net and MobileNetV2 models. The proposed approach holds the importance to meet the needs of applications requiring more complex human-computer interaction systems Kopuklu et al. (2019). A CNN-based gesture recognition method was proposed with different procedures. The procedures involve the implementation of morphological filters, contour generation, polygonal approximation, and segmentation during preprocessing Pinto et al. (2019). This process ensures a better feature extraction. Based on the training and testing, the proposed procedural CNN performs well in terms of metrics and convergence graphs. From the literature, it is evident that CNN is the most popular technique for deep learning. In this paper, we paid attention to develop a web application (app) for hand cricket using CNN.

3. Proposed Web Application

The proposed web application framework is presented in detail in this section. CNN model is discussed briefly; the framework follows later.

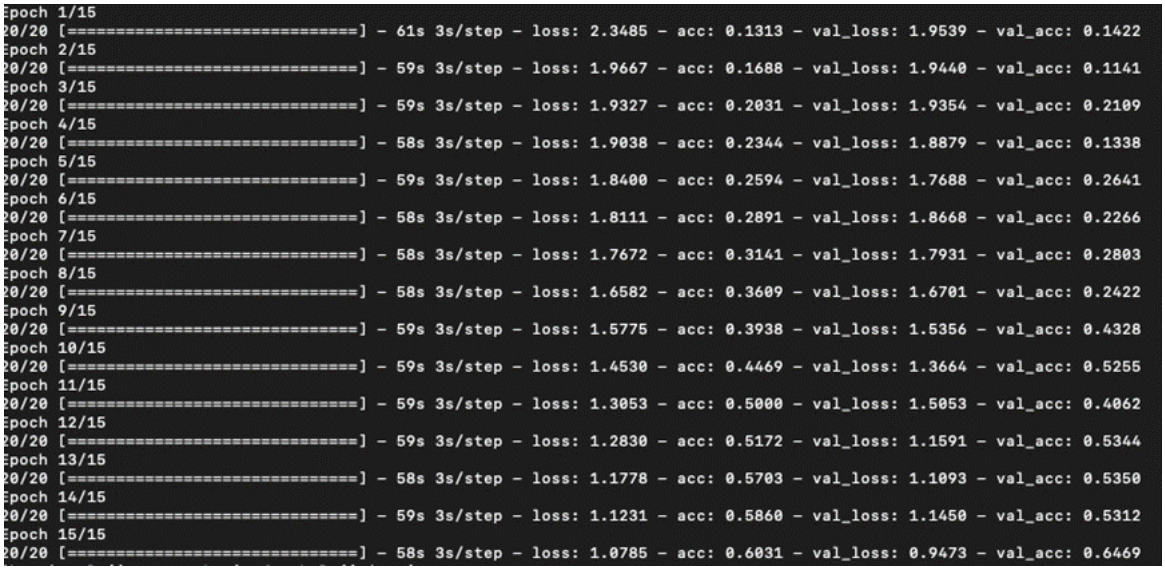
Epoch 1/15
20/20 [==============================] - 61s 3s/step - loss: 2.3485 - acc: 0.1313 - val_loss: 1.9539 - val_acc: 0.1422
Epoch 2/15
20/20 [==============================] - 59s 3s/step - loss: 1.9667 - acc: 0.1688 - val_loss: 1.9440 - val_acc: 0.1141
Epoch 3/15
20/20 [==============================] - 59s 3s/step - loss: 1.9327 - acc: 0.2031 - val_loss: 1.9354 - val_acc: 0.2109
Epoch 4/15
20/20 [==============================] - 58s 3s/step - loss: 1.9038 - acc: 0.2344 - val_loss: 1.8879 - val_acc: 0.1338
Epoch 5/15
20/20 [==============================] - 59s 3s/step - loss: 1.8400 - acc: 0.2594 - val_loss: 1.7688 - val_acc: 0.2641
Epoch 6/15
20/20 [==============================] - 58s 3s/step - loss: 1.8111 - acc: 0.2891 - val_loss: 1.8668 - val_acc: 0.2266
Epoch 7/15
20/20 [==============================] - 58s 3s/step - loss: 1.7672 - acc: 0.3141 - val_loss: 1.7931 - val_acc: 0.2803
Epoch 8/15
20/20 [==============================] - 58s 3s/step - loss: 1.6582 - acc: 0.3609 - val_loss: 1.6701 - val_acc: 0.2422
Epoch 9/15
20/20 [==============================] - 59s 3s/step - loss: 1.5775 - acc: 0.3938 - val_loss: 1.5356 - val_acc: 0.4328
Epoch 10/15
20/20 [==============================] - 59s 3s/step - loss: 1.4530 - acc: 0.4469 - val_loss: 1.3664 - val_acc: 0.5255
Epoch 11/15
20/20 [==============================] - 59s 3s/step - loss: 1.3053 - acc: 0.5000 - val_loss: 1.5053 - val_acc: 0.4062
Epoch 12/15
20/20 [==============================] - 59s 3s/step - loss: 1.2830 - acc: 0.5172 - val_loss: 1.1591 - val_acc: 0.5344
Epoch 13/15
20/20 [==============================] - 58s 3s/step - loss: 1.1778 - acc: 0.5703 - val_loss: 1.1093 - val_acc: 0.5350
Epoch 14/15
20/20 [==============================] - 59s 3s/step - loss: 1.1231 - acc: 0.5860 - val_loss: 1.1450 - val_acc: 0.5312
Epoch 15/15
20/20 [==============================] - 58s 3s/step - loss: 1.0785 - acc: 0.6031 - val_loss: 0.9473 - val_acc: 0.6469

Figure 4. Training Logs

3.1. CNN Model

Convolutional Neural Networks (CNN) is a class of deep learning networks and is most widely used for analyzing videos and images. Fundamentally, CNN has input layers, multiple hidden layers, and the output layer. The simplest net training called backpropagation is applied for verification of the recognized gestures and various human–computer interactions and provides better accuracy and efficiency. In the application of image analysis, it is significantly used for image classification, object recognition, and detection. Figure 2 shows the summarized form of CNN layers in its structure: convolution, pooling and classification. The architecture of CNN is defined based on the following: application, number of neurons in each layer, the activation function, and the number of alternate convolution and pooling layers Trigueiros et al. (2012). For image classification, the arbitrary colour model is used to represent an image and given as input for a CNN. In the convolution layer, each neuron relates to a kernel window which is convolved with the input image while training and classification of CNN. Each associated neuron is composed of weights. The convolution step output is a set of N images, and each image is associated with N neurons. The convolved new images may contain negative values. To avoid such issues, the negative values are replaced with zero using a Rectified Linear Unit(ReLU). This layer output is called feature maps. The output of the convolution layer is the input for the pooling layer. The pooling layer reduces the dimension of feature maps which reduces the network training time. Certain architectures alternate between the convolution and pooling layers. The final layer is a multiplayer perceptron neural network. It does the feature classification maps computed by the intermediate convolution and pooling layers.

In the proposed app development, the CNN method is used to identify human hand gestures. To realize robust performance, the skin model and the hand position alignment and orientation are put in to obtain the required data for training and testing. The calibration and orientation of hand position aim at translating and rotating the hand image to a neutral pose. Later CNN is trained using the calibrated images. In this work, the proposed method is validated by observing the human gestures that show robust results with various hand positions, orientations, and light conditions.

3.2. Software Framework

The entire application is developed using the language Python and is implemented in a variety of text editors and IDEs. First, a model is built and trained. The trained model is later deployed in an OpenCV (Open-Source Computer Vision Library) application which will do the real-time

Computer vision task of detecting gestures and returning those values instantaneously.

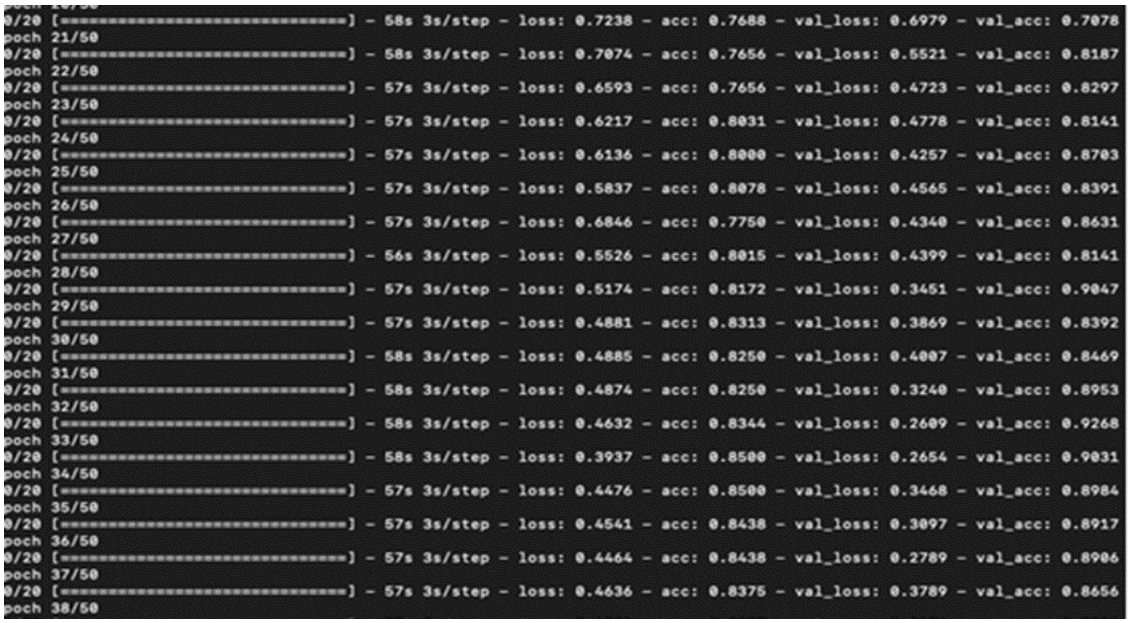

Figure 5. Training logs with better accuracy

3.2.1. Dataset

Finding a dataset is the biggest problem in deep learning to train a perfect model for prediction tasks. However, the available one for free is the sign language for the alphabet A and numbers 0-9 with limited datasets. Therefore, to overcome that additional 200 pictures in each class that is captured manually in different backgrounds are added. As the batch size and the number of epochs were small, and a lot of training data was not utilized properly, resulted in low accuracy. To counter this, the batch size and the number of epochs were increased, and add callbacks to stop training after reaching a satisfying accuracy. Totally 1400 images were taken for training. All these images were captured from different age group on different lighting conditions, backgrounds and ambiance. A sample of the dataset used is shown in Fig. 3.

3.2.2. Loading Data

To load the data, we extensively made use of the Keras class "ImageDataGenerator" to load and augment the images. The image classification is done with the help of ImageDataGenerator. The ImageDataGenerator can be used in several ways, depending on the method we adopt. This work was focused on flow_from_directory. It draws a pathway to the directory containing images. The images are sorted in subdirectories and image augmentation parameters. All the images are saved in ordered directories. In addition, the method of ImageDataGenerator is used to load the train and for the validation of each class separately. In addition to that, while the images are being sent for training, they are augmented simultaneously in a variety of ways like skewing, lateral inversions, zoomings, cropping, rotations, etc. These inputs are given by adding parameters to the flow_from_directory method of the ImageDataGenerator. The augmentation does not take any extra time or extra space to augment and load.

3.2.3. Creating Model

An input image is fed into a CNN-based Deep Learning algorithm. The filters are assigned to various aspects of the image and could differentiate one from the other. In CNN the preprocessing requirement is much lower as compared to other classification algorithms. Instead of trying to

handcraft features that can help with classification with enough training, CNNs can learn these automatically as weights. A CNN like any artificial neural network (ANN) can have any network architecture, instead of creating network architecture from scratch, transfer learning is applied by importing an existing CNN called the SqueezeNet. SqueezeNet is employed due to its minimal memory requirements and rapid processing speed. An extra softmax layer is added to the SqueezeNet to get the exact number of output classes required for the game. Squeeznet architecture is chosen for its capability of providing a balance between accuracy and computational resources. More specifically the power to train and deploy. Also, squeezed was designed with fewer parameters and it is easily transmitted over network.

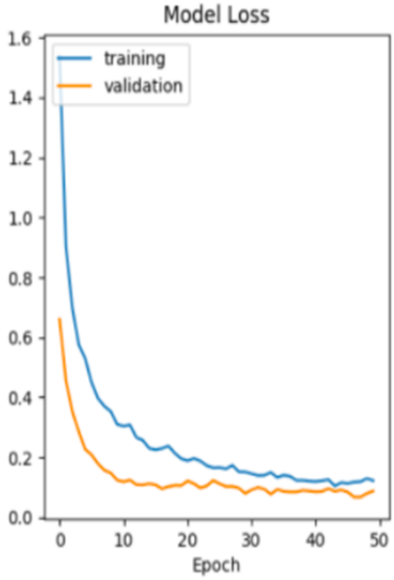

Figure 6. Model loss vs Number of Epochs

3.2.4. Training Model

Once the model is built it must be trained to get the perfect weights. For training, the Categorical cross-entropy loss function was used. We have 7 output classes and use the Adam optimizer to help facilitate fast and efficient learning. These inputs are given in the model, compile () method. Then the model is trained using the model, fit () method. The parameters like the number of epochs, batch size, and callbacks are specified in the method. The number of epochs is a hyper parameter in Machine learning. It defines the number of times that the learning algorithm iterates through the entire training dataset. Every sample in the training dataset enables in updating internal model parameters is denoted as an epoch. An epoch comprises one or more batches. To train the classifier for the game, 15 epochs are set, and the training is called back when the validation accuracy reaches 95%. It is done to avoid any overfitting. Overfitting occurs when the model absorbs the detail and learns about the noise in the training data. This in turn negatively impacts the performance of the model on new data. Also, the noise variations in the training data are picked up and learned as concepts by the model. The issue is that new data does not make use of these concepts. Thus, for the model to make accurate predictions all the above steps are taken. Once the training is done, the CNN weights are stored as a .h5 file .

3.2.5. Testing Model

The trained model is imported and using the model, predict () method in Keras, we can test on the images in the validation directory. These help in giving the unbiased accuracy of our classifier. Later, a confusion matrix is produced, which is a specific table layout that allows visualization of the performance of an algorithm. Certain aspects of our problem are directly related to the accuracy of

our model. Reasonably distinct and clear images were presented without background. The number of images is quite reasonable, which makes the chosen model more robust. The shortcoming is that for various problems, apparently more data would be needed to stir the parameters of the chosen model in a better direction. Also, a learning model is very rigid to interpret, given its abstractions. However, it is much easier to work on the actual issue using this approach, since we do not have to account for feature engineering. Hence, preprocessing of the images is not required to extract the important features. CNN does feature extraction. This also aids in adapting to new problems relatively easily. As specified, the alternative approach to this problem would be to use feature engineering, such as binary thresholding (check area of the hand), circle detection, and others to detect unique characteristics of the images. However, with the CNN approach, there is no need to worry about any of these.

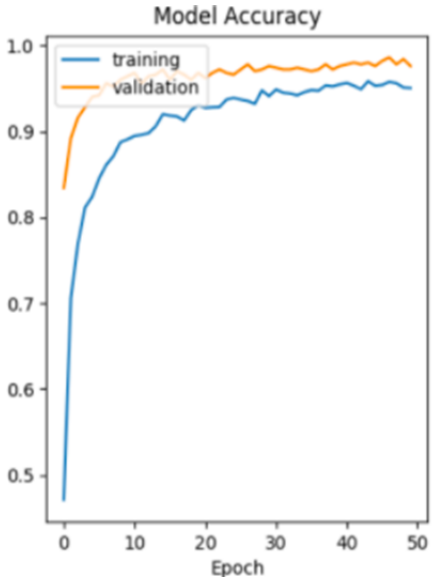

Figure 7. Training and validation accuracy plot

Figure 8. The user and the game app interface

4. Building the Game Application

OpenCV2 library is imported along with the required TensorFlow libraries to deploy the trained model. A key-value pair dictionary is created to map the 7 class values to numbers. An OpenCV

window is opened when the user opens the .exe game. It is done using the CreateWindow method. For the toss, the user has to enter 't' and it is read using the Waitley method. A variable prev is assigned with the value 0 which translates to none. Two rectangular boxes are created, one for getting the input and the other box generating random images that show hand signs from 1 to 6. The game is written such that whenever a user changes the hand sign and it is not equal to the value of prev, a random move by the computer is also made. By using a few if-else instances the game rules are written. The scores are added in two variables named user and computer. The user stores the runs the user made and the comp variable stores the runs the computer made. Once both players (user and computer) are out, user and comp values are compared, and the winner is announced. To reset the game, the user should press r. It is done by applying the Waitley method.

5. Results and Discussions

The user experiments on the detection accuracy of the developed prototype, detecting correctly all the six hand gestures made on either hand, in a controlled environment have been reported.

5.1. Training Accuracy Logs with few Epochs

When the batch size and the number of epochs were small, a lot of training data was not utilized properly hence we received a low accuracy of only 65%. The training logs with few epochs are given in Fig. 4.

5.2. Accuracy Logs with the increased number of Epochs

To increase this, the batch size and the number of epochs are increased, and add callbacks to stop training after reaching 93% accuracy. The model loss and accuracy as functions of the epochs are plotted immediately after training by the plot method from the numpy library in python. The accuracy logs with the increased number of epochs are seen in Fig. 5.

5.3. Model vs Number of Epochs Plot

Figure 6 depicts the relation between model loss versus the number of epochs. The loss as a function of the epochs is plotted using the numpy method plot. The plot function takes the training logs as its parameters for the validation as mentioned earlier. From the graph, it is observed that as the number of epochs raises, the loss incurred by the considered model is reduced. The plot is drawn for both the training case and the validation of the model. The responses of both curves are very similar.

5.4. The training and validation accuracy plot

The training and validation accuracies as functions of epochs are plotted in Fig. 7. The training logs are given as the input parameter for the plot function. From the plot, it is clearly shown that the accuracy is maximum as the number of epochs increases. The training and validation curves for the given dataset perform alike for the model accuracy.

5.5. Game App Interface

The game user plays the game by showing a number as shown in Fig. 8. In that instance, the user shows the number 4, which gets added up to the total runs variable displayed there. The computer's move is 2 and it is displayed in the box to the right. As both the values were different the user value was added to the total.

6. Conclusions

In this paper, a method for playing the hand cricket game was presented by identifying hand gestures based on computer-vision techniques. The developed application works together with a

real-time implementation on an ordinary webcam. Several user experiments were carried out for the accurate analysis of the developed prototype. Detecting all the six hand gestures correctly was made on either hand, in a controlled environment. Proposed Fast SqueezeNet outperforms well concerning the parameters, reasoning time and portability of the model, when compared with that of other models. Thus, an interactive simple hand cricket game was developed. The same application development can be further tried using recurrent neural network algorithms.

References

B., A; Krishi, K; M., M; Daaniyaal, M, and S., A. H. Hand gesture recognition using machine learning algorithms. *Computer Science and Information Technologies*, 1, 2020. ISSN 2722-323X. doi: 10.11591/csit.v1i3.p116-120.

Chen, Z. H; Kim, J. T; Liang, J; Zhang, J, and Yuan, Y. B. Real-time hand gesture recognition using finger segmentation. *Scientific World Journal*, 2014, 2014. ISSN 1537744X. doi: 10.1155/2014/267872.

Kingma, D. P and Ba, J. L. Adam: A method for stochastic optimization. 2015.

Kopuklu, O; Rong, Y, and Rigoll, G. Talking with your hands: Scaling hand gestures and recognition with cnns. 2019. doi: 10.1109/ICCVW.2019.00345.

Munasinghe, M. I. Dynamic hand gesture recognition using computer vision and neural networks. 2018. doi: 10.1109/I2CT.2018.8529335.

Pinto, R. F; Borges, C. D; Almeida, A. M, and Paula, I. C. Static hand gesture recognition based on convolutional neural networks. *Journal of Electrical and Computer Engineering*, 2019, 2019. ISSN 20900155. doi: 10.1155/2019/4167890.

Prakash, J and Gautam, U. K. Hand gesture recognition. *International Journal of Recent Technology and Engineering*, 7, 2019. ISSN 22773878. doi: 10.22214/ijraset.2023.48557.

Sudeep, K. S and Pal, K. K. Preprocessing for image classification by convolutional neural networks. 2017. doi: 10.1109/RTEICT.2016.7808140.

Trigueiros, P; Ribeiro, F, and Reis, L. P. A comparison of machine learning algorithms applied to hand gesture recognition. 2012.

Triviño-López, I. C; Rodríguez-Garavito, C. H, and Martinez-Caldas, J. S. Hand gesture recognition using computer vision applied to colombian sign language. volume 12014 LNCS, 2020. doi: 10.1007/978-3-030-45096-0_26.